Performance of various BFGS and DFP implementations with limited precision second order information

D. Byatt, I. D. Coope and C. J. Price

Department of Mathematics and Statistics
University of Canterbury
Private Bag 4800
Christchurch, New Zealand

Report Number: UCDMS2003/1 January 2003

Keywords: Quasi-Newton methods, BFGS, DFP, numerical stability.

PERFORMANCE OF VARIOUS BFGS AND DFP IMPLEMENTATIONS WITH LIMITED PRECISION SECOND ORDER INFORMATION

D. BYATT, I. D. COOPE, AND C. J. PRICE

ABSTRACT. This paper supports the claim that there is no numerical advantage in choosing factorised implementations (over non-factorised implementations) of BFGS or DFP quasi-Newton methods when approximate Hessian information is available to full machine precision. However the results presented in this paper show that a factorisation strategy has clear advantages when approximate Hessian information is available only to limited precision. These results show that a conjugate directions factorisation outperforms all other methods considered in this paper (including Cholesky factorisation) for both BFGS and DFP formulae.

1. INTRODUCTION

Quasi-Newton algorithms are used to solve the local optimisation problem

$$\min_{x \in \mathbb{R}^n} f(x)$$

iteratively, where $f\colon \mathbb{R}^n \to \mathbb{R}$ and gradient information is available. The solution is attained when $\nabla f(x) = 0$, but in practice the usual requirement is that $\|\nabla f(x)\| \leqslant \tau_g$ for some (typically small) positive constant τ_g.

The development of quasi-Newton, or variable metric algorithms, as they were originally called is attributed to Davidon in 1959 [8] and became popularised as the "DFP" method by Fletcher and Powell in 1963 [13]. This method was found to work well in practice when used in conjunction with accurate line searches. However the DFP method is less effective when used with the low accuracy line searches which have become popular since the 1970s.

Work by Broyden [2, 3, 4], Fletcher [11], Goldfarb [17], Shanno [27], and also Greenstadt [20], led to the development of the "BFGS"

Date: January 2003.

This research was financially supported by a Top Achiever Doctoral Scholarship.

method. In practice BFGS outperforms DFP when used with low accuracy line searches, even though both produce identical iterates with exact arithmetic — a remarkable result shown by Dixon in 1972 [10, 11].

There are many alternative quasi-Newton update formulae, for example, "SR1" the symmetric rank one update and the (infinitely large) Broyden family of updates (of which BFGS and DFP are members).

Throughout this paper the convention of writing $f(x_k)$ as f_k and $\nabla f(x_k)$ as g_k is used. At iteration k of a quasi-Newton method a search direction p_k is found by solving the system of equations

$$B_k p_k = -g_k \tag{1}$$

where B_k approximates, in some sense, the Hessian matrix $\nabla^2 f(x_k)$. A line search is then performed along $x_k + \alpha p_k$, $\alpha \in \mathbb{R}$ to find a new iterate $x_{k+1} = x_k + \alpha_k p_k$ for some α_k that satisfies the line search criteria. Information at this new point is used to generate a new approximate Hessian matrix B_{k+1}.

If B_k is positive definite then $p_k^T g_k < 0$ so that p_k is a descent direction for f. In this situation the line search is typically replaced by a ray search ($\alpha > 0$).

The use of Cholesky factorisations of the approximate Hessian matrices B_k was introduced in [15] and is now in widespread use (it is coded as `VA13A` in the Harwell subroutine library [22], for example). Proponents of this implementation claim it avoids the computational instability of using the inverses of the approximate Hessian matrices and allows the efficient calculation of the search direction in $O(n^2)$ operations by using both forward and back substitution. The standard Cholesky factorisation implementation of the BFGS method uses the *modified* Cholesky factorisation $B_k = L_k D_k L_k^T$ where L_k is unit lower triangular and D_k is diagonal. The modified implementation allows the easy detection (and subsequent correction) of loss of positive definiteness of the approximate Hessian matrices (due to rounding errors in finite precision arithmetic) with little extra computational effort. As the theory of Cholesky factorisations is well established (see for example, [1, 16, 27] and the references contained therein) it is not discussed further here.

This paper investigates the performance of 22 BFGS and DFP implementations on a selection of ill-conditioned test problems across a range of dimensions and line search criteria. The results presented in this paper support those in [19, 20], specifically that:

- There is no numerical evidence to support the claim that a Cholesky factor implementation of the BFGS formula offers any

improvement in performance, as is popularly believed, over more straightforward implementations when second order information is available to full precision.

- The numerical instability of non-factored implementations of quasi-Newton methods reported by some authors is due to early implementations of the DFP formula with low accuracy line searches.

and extend these results to show that:

- A factorisation strategy has clear advantages when second order information is only available to limited precision. However a Cholesky factorisation is not necessarily the best one to use.

2. BFGS AND DFP FORMULAE

The BFGS and DFP update formulae can be written as

$$\text{(BFGS)} \quad B_{k+1} = \left[B + \frac{yy^{\mathsf{T}}}{s^{\mathsf{T}}y} - \frac{Bss^{\mathsf{T}}B}{s^{\mathsf{T}}Bs} \right]_k \tag{2}$$

and

$$\text{(DFP)} \quad B_{k+1} = \left[B + \left(1 + \frac{s^{\mathsf{T}}Bs}{s^{\mathsf{T}}y}\right) \frac{yy^{\mathsf{T}}}{s^{\mathsf{T}}y} - \left(\frac{ys^{\mathsf{T}}B + Bsy^{\mathsf{T}}}{s^{\mathsf{T}}y} \right) \right]_k \tag{3}$$

where $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$. If the inverse of B_k is denoted by H_k then application of the Sherman-Morrison-Woodbury formula [25, 29, 30] gives

$$\text{(BFGS)} \quad H_{k+1} = \left[H + \left(1 + \frac{y^{\mathsf{T}}Hy}{s^{\mathsf{T}}y}\right) \frac{ss^{\mathsf{T}}}{s^{\mathsf{T}}y} - \left(\frac{sy^{\mathsf{T}}H + Hys^{\mathsf{T}}}{s^{\mathsf{T}}y} \right) \right]_k \tag{4}$$

and

$$\text{(DFP)} \quad H_{k+1} = \left[H + \frac{ss^{\mathsf{T}}}{s^{\mathsf{T}}y} - \frac{Hyy^{\mathsf{T}}H}{y^{\mathsf{T}}Hy} \right]_k. \tag{5}$$

Equations (4) and (5) allow the direct calculation of the search direction without the need to solve the system of equations (1).

Each of the implementations discussed in this paper fall into three general categories:

- Direct updates of the approximate Hessian matrices using equation (2) for the BFGS methods and equation (3) for the DFP methods with various methods of solving the resulting system of equations.

- Direct updates of the inverses of the approximate Hessian matrices using equation (4) for the BFGS methods and equation (5) for the DFP methods.
- Factorisations: Either Cholesky factorisations of the approximate Hessian matrices, or conjugate factorisations of their inverses.

The method of conjugate factorisation used in this paper is based on [7]. A brief description is given in the following section, but see [7] for more details.

2.1. **Conjugate factorisation.** The BFGS update formula (4) can be written in product form [1] as

$$H_{k+1} = \left[(I - pq^{\mathsf{T}}) H (I - pq^{\mathsf{T}})^{\mathsf{T}} \right]_k$$

where

$$q_k = \left[\frac{y}{p^{\mathsf{T}} y} \pm \frac{g}{\sqrt{-p^{\mathsf{T}} g \, p^{\mathsf{T}} y / \alpha}} \right]_k.$$

If the inverse Hessian approximation matrices are factored so that $H_k = C_k C_k^{\mathsf{T}}$ then the columns of C_k are B_k-conjugate and the search direction is given by $p_k = -C_k d_k$ where $d_k = C_k^{\mathsf{T}} g_k$ are the directional derivatives of f at x_k in the directions of the columns of C_k. The updated conjugate factors can be written as

$$C_{k+1} = \left[C - \frac{p z^{\mathsf{T}}}{p^{\mathsf{T}} y} \mp \frac{p d^{\mathsf{T}}}{\sqrt{-p^{\mathsf{T}} g \, p^{\mathsf{T}} y \, / \alpha}} \right]_k \tag{6}$$

where $z_k = C_k^{\mathsf{T}} y_k$ is the difference between the directional derivatives at x_{k+1} and x_k. Then $d_{k+1} = C_{k+1}^{\mathsf{T}} g_{k+1}$ can be written as

$$d_{k+1} = \bar{d}_k - \frac{p_k^{\mathsf{T}} g_{k+1} \, z_k}{p_k^{\mathsf{T}} y_k} \mp \frac{p_k^{\mathsf{T}} g_{k+1} \, d_k}{\sqrt{-p_k^{\mathsf{T}} g_k \, p_k^{\mathsf{T}} y_k \, / \alpha_k}} \tag{7}$$

where $\bar{d}_k = C_k^{\mathsf{T}} g_{k+1}$. Equations (6) and (7) can be written in terms of the new variables d and z so that

$$C_{k+1} = \left[C + \frac{p z^{\mathsf{T}}}{d^{\mathsf{T}} z} \mp \frac{p d^{\mathsf{T}}}{\sqrt{-d^{\mathsf{T}} d \, d^{\mathsf{T}} z \, / \alpha}} \right]_k$$

and

$$d_{k+1} = \left[\bar{d} - \frac{d^{\mathsf{T}} \bar{d} \, z}{d^{\mathsf{T}} z} \pm \frac{d^{\mathsf{T}} \bar{d} \, d}{\sqrt{-d^{\mathsf{T}} d \, d^{\mathsf{T}} z \, / \alpha}} \right]_k.$$

There are two obvious implementations, one for each of the $+/-$ signs in equation (6). These are denoted by the letters "p" and "m" in the following sections.

2.2. **Implementations.** There are many ways to implement the BFGS and DFP formulae presented in equations (2)–(5). The names given to the methods discussed in this paper are prefixed in a natural way so that those prefixed with "B" use the BFGS formula and those prefixed with "D" use the DFP formula. As Matlab [23] was used to produce all numerical results, Matlab's built-in functions were used where convenient. Details of each of the implementations considered in this paper are now presented. Text in `typewriter` font is used to emphasize Matlab code.

Bihess. Inverse BFGS formula and direct calculation of the search direction. Uses equation (4) to update the sequence of H_k matrices. The search direction is calculated directly via $\mathtt{p_k} = -\mathtt{H_k} * \mathtt{g_k}$.

Binv. BFGS formula and matrix inverse. Uses equation (2) to update the sequence of B_k matrices. The search direction is calculated by using the Matlab matrix inverse function via the equation $\mathtt{p_k} = -\mathtt{inv}(\mathtt{B_k}) * \mathtt{g_k}$. Note that this method of solving the system of equations (1) is never recommended in practice as it is more computationally expensive and considered to be less numerically stable than solving the system of equations by other means. It is used here to provide a guideline for the worst performance that would be expected from this type of implementation.

Bgauss. BFGS formula and Gaussian elimination. Uses equation (2) to update the sequence of B_k matrices. The search direction is calculated by solving equation (1) with Gaussian elimination by using Matlab's "backslash" command via $\mathtt{p_k} = -\mathtt{B_k} \backslash \mathtt{g_k}$.

Bgaussg. Essentially the same method as Bgauss except that the BFGS formula suggested in [17, p. 119] and reproduced as equation (8) is used. Since all quasi-Newton methods are based on the equation $B_k p_k = -g_k$, and $s_k = \alpha_k p_k$ it follows that BFGS update equation (2) can be written as

$$B_{k+1} = \left[B + \frac{yy^{\mathsf{T}}}{s^{\mathsf{T}}y} + \frac{gg^{\mathsf{T}}}{p^{\mathsf{T}}g} \right]_k. \tag{8}$$

Bcholu. BFGS formula and Cholesky factorisation of the B_k matrices. Uses a sequence of Cholesky factors L_k which are updated (rather than recomputed from scratch) at each iteration. The particular implementation presented here uses Matlab's Cholesky factor update command `cholupdate`. The search direction is calculated with forward and back substitution via $\mathtt{p_k} = -\mathtt{L_k^T} \backslash (\mathtt{L_k} \backslash \mathtt{g_k})$.

Bcholug. Essentially the same method as Bcholu except that equation (8) is used to update the approximate Hessian information.

Bconjp. BFGS formula and conjugate factorisation of the inverse approximate Hessian matrices using the plus sign from equation (6).

Bconjpt. Essentially the same implementation as Bconjp except that the conjugate factors are triangularised by using a QR factorisation via the Matlab `qr` command.

Bconjptu. Essentially the same implementation as Bconjpt except that the triangular factors are updated at each iteration (with the Matlab command `qrupdate`) rather than recomputed from scratch.

Bconjm, Bconjmt, Bconjmtu. The same implementations as Bconjp, Bconjpt and Bconjptu except that the minus sign from equation (6) is used.

DFP implementations. Each of the DFP implementations is the DFP equivalent of one of the BFGS implementations described above with the exception that there is no DFP equivalent for Bgaussg or Bcholug.

2.3. **Practicalities.** The implementations Binv, Bgauss and Bgaussg (and their DFP counterparts Dinv and Dgauss) require $O(n^3)$ operations at each iteration to update the second order information and compute the new search direction, whereas the remaining implementations require only $O(n^2)$ operations. Additionally, the (modified) Cholesky factorisation and triangular conjugate factorisation implementations allow the easy detection of loss of positive definiteness of the approximate Hessian matrices. The other implementations do not have this feature. However with a conjugate factorisation it is extremely unlikely that the inverse approximate Hessian matrices will lose positive definiteness. The worst that can happen is that they may become positive semi-definite. In fact Powell makes the comment in [26] that:

> We even find that, if we let Z [the conjugate factorisation matrix] be singular initially, then in practice the rounding errors of a sequence of updating calculations remove the singularity very successfully.

Thus if positive definiteness of the inverse approximate Hessian matrices is lost then it is extremely likely it will be restored at the next iteration — or the other way around — it is extremely unlikely that any loss of definiteness will be maintained for any length of time if conjugate factors are used.

Function	*Dim.*	*Initial point*	*Min.*
Rosenbrock	2	$(-1.2, 1)$	0
Powell badly scaled	2	$(0, 1)$	0
Repeated Rosenbrock	4	$(-1.2, 1, -1.2, 1)$	0
Multi-dimensional Rosenbrock	4	$(-1.2, 1, -1.2, 1)$	0
Powell singular	4	$(3, -1, 0, 1)$	0

Table 1. Low dimension test functions.

Function	*Initial point*	*Min.*
Repeated Rosenbrock	$(-1.2, 1, -1.2, 1, \ldots)$	0
Multi-dimensional Rosenbrock	$(-1.2, 1, -1.2, 1, \ldots)$	0
Powell singular	$(3, -1, 0, 1, \ldots)$	0
Hilbert quadratic	$(0, 0, 0, 0, \ldots)$	0

Table 2. Test functions for 8, 12, 20, 40 and 60 dimensions.

3. Numerical results

Each of the 22 quasi-Newton implementations described above were tested with two different line searches on the suite of 25 test functions listed in Tables 1 and 2 as the precision of the approximate Hessian information varied from 16 to two digits. The varying levels of precision were achieved by truncating the elements of the approximate Hessian matrices (possibly in factored form, or their inverses) to the desired level. For example, the elements of the matrix X are truncated to n digits with $\text{trunc}(X) = 10^{-d}\lfloor 10^d X \rfloor$ where $d = n - \lceil \log_{10}(\max(|X|)) \rceil$.

Each of the higher dimensional tests listed in Table 2 were carried out in 8, 12, 20, 40 and 60 dimensions. More details on the test functions can be found in [19, 20, 24].

A strong Wolfe line search was used so that at each iteration α_k was chosen so that $x_{k+1} = x_k + \alpha_k p_k$ satisfies

$$f_{k+1} \leqslant f_k + \rho \alpha_k p_k^{\mathsf{T}} g_k$$

and

$$|p_k^{\mathsf{T}} g_{k+1}| \leqslant \sigma |p_k^{\mathsf{T}} g_k|$$

where the sufficient descent parameter $\rho = 10^{-4}$ and the gradient parameter σ was set to 10^{-3} and 0.9 for what are referred to in the remainder of this paper as *strong* and *weak* line searches. The Wolfe line search was implemented using a safeguarded parabolic interpolation scheme.

For each test problem the number of function evaluations, final function value and execution time (in seconds) were recorded. The overall

performance of each implementation was determined using the following ranking system. Firstly the implementations were sorted by the number of test functions that were successfully solved. A test problem was deemed to have been successfully solved if the termination criterion $\|\nabla f(x)\| \leqslant 10^{-6}$ was met. If necessary the algorithms were then subsorted by the mean number of function evaluations. Any ties were subsorted by the mean accuracy of the approximations to the minimum function values. The accuracy was measured using $\log_{10}(f - f^*)$ where f^* represents the minimum of the function and f is the final function value, see [5, p. 60] for more details. Note that only data for the problems that were solved successfully were used in the sorting process.

As it is the "raw" performance of each implementation that is being investigated, the algorithms were terminated whenever they ran into difficulty rather than applying some sort of corrective procedure. For example, if a descent direction is not found (implying the loss of positive definiteness of the current Hessian approximation), the algorithm is terminated even though corrective procedures are available. The implementations were deemed unsuccessful and thus terminated if:

- The line search failed.
- A descent direction was not found.
- A factorisation failed (where appropriate).
- More than 10^5 function evaluations were required.

As algorithm execution time depends on the computing environment as well as the implementation, the mean execution times presented here (although not used in the ranking scheme) should only be considered as an indication of the relative time required by each algorithm. All of the implementations presented in this paper were run in a Matlab R12.1 environment on a Sun-Fire-880 multi-user machine with four 750MHz processors and 8GB of RAM running Solaris 8.

Note also that only the data for the test functions that were solved successfully are presented. In each of the following results tables the columns labelled *Succ*, *Fcnt*, *Accy* and *Time* represent the number of successfully solved test problems, the mean number of function evaluations, the mean accuracy of the solutions and the mean execution time in seconds.

As can be seen from the results presented below, when successful, all implementations produced similarly accurate approximations to the solutions of the test problems, but the BFGS implementations tended to required fewer function evaluations than the DFP implementations. Furthermore, the number of function evaluations required by

Rank	*Method*	*Succ*	*Fcnt*	*Accy*	*Time*
1	Bihess	25	208.6	-14.2	0.5
2	Bconjpt	25	215.9	-14.5	0.6
3	Bconjptu	25	216.1	-14.3	0.6
4	Bconjm	25	216.2	-14.6	0.5
5	Bconjmt	25	218.6	-14.4	0.6
6	Bcholu	25	219.2	-14.1	0.6
7	Bcholug	25	219.3	-14.3	0.6
8	Bconjp	25	219.9	-14.3	0.6
9	Bconjmtu	25	222.9	-14.5	0.6
10	Bgaussg	25	229.8	-13.8	0.6
11	Bgauss	25	239.4	-14.2	0.6
12	Binv	25	241.2	-14.6	0.7
1	Dihess	25	238.4	-14.5	0.6
2	Dconjp	25	240.8	-14.5	0.6
3	Dconjm	25	249.7	-14.3	0.7
4	Dconjmtu	25	249.8	-14.0	0.7
5	Dconjptu	25	250.9	-13.9	0.7
6	Dconjmt	25	254.3	-14.4	0.7
7	Dconjpt	25	255.4	-14.6	0.7
8	Dcholu	25	255.7	-14.4	0.7
9	Dgauss	25	264.1	-14.7	0.7
10	Dinv	25	269.9	-14.5	0.7

Table 3. Strong line search and 16 digit second order information.

the DFP implementations increased dramatically with the weak line search — as expected due to the known instability of DFP methods with low accuracy line searches.

3.1. **Full precision second order information.** The performance of each implementation with full precision (16 digits) second order information for the strong and weak line searches is discussed in the following sections. The results are presented in Tables 3 and 4.

Strong line search. All of the 22 implementations solved all 25 test problems. The mean number of function evaluations ranged from 208.6 for Bihess through to 269.9 for Dinv. The mean number of function evaluations required by the BFGS implementations was 225 ± 17 compared to 254 ± 16 for the DFP implementations. The mean execution times ranged from 0.5 to 0.7 seconds per test problem. Overall all BFGS implementations produced very similar results, as did all DFP implementations.

Weak line search. The mean number of function evaluations ranged from 110.3 for Bcholug through to 21666.7 for Dcholu. The mean

Rank	*Method*	*Succ*	*Fcnt*	*Accy*	*Time*
1	Bcholug	25	110.3	-10.4	0.3
2	Bcholu	25	111.7	-10.4	0.3
3	Bgaussg	25	112.8	-10.8	0.3
4	Binv	25	112.8	-10.6	0.4
5	Bconjm	25	113.0	-10.5	0.3
6	Bconjp	25	113.9	-10.5	0.3
7	Bgauss	25	115.0	-10.8	0.4
8	Bconjpt	24	114.8	-10.8	0.4
9	Bconjmt	24	114.9	-10.9	0.4
10	Bconjptu	24	115.2	-10.8	0.4
11	Bihess	24	116.3	-11.0	0.3
12	Bconjmtu	24	116.8	-11.0	0.4
1	Dcholu	24	21666.7	-10.3	79.1
2	Dihess	22	16406.4	-9.7	50.8
3	Dconjp	21	11284.6	-10.2	22.1
4	Dgauss	21	16003.1	-11.1	37.4
5	Dconjm	20	12228.4	-10.3	26.8
6	Dinv	20	12497.5	-10.2	40.5
7	Dconjptu	19	9490.0	-10.0	25.2
8	Dconjmtu	19	11582.1	-9.7	31.7
9	Dconjmt	18	8990.3	-10.8	31.0
10	Dconjpt	18	11783.6	-10.6	40.2

Table 4. Weak line search and 16 digit second order information.

number of function evaluations required by the BFGS implementations was 114 ± 4 compared to 15330 ± 6340 for the DFP implementations. The big difference in the mean number of function evaluations was also reflected in the mean execution times which ranged from 0.3 to 79.1 seconds per test problem. With the weak line search the BFGS implementations required far fewer function evaluations than the DFP implementations. Although the BFGS implementations with the strong line search were slightly more robust than the BFGS implementations with the weak line search they required nearly double the number of function evaluations. Although highly dependent on the line search, this is a major reason for the popularity of weak line searches.

3.2. **Limited precision second order information.** The performance of each implementation as the precision of the second order information varied from 16 to two digits with the strong and weak line searches is discussed in the following sections. The results are presented in Tables 5 and 6.

Rank	*Method*	*Succ*	*Fcnt*	*Accy*	*Time*
1	Bconjm	334	216.4	-14.2	0.6
2	Bconjp	334	222.6	-14.1	0.6
3	Bconjptu	327	244.0	-14.1	0.7
4	Bconjmt	327	245.2	-14.1	0.7
5	Bconjpt	327	247.4	-14.2	0.7
6	Bconjmtu	326	243.3	-14.1	0.7
7	Bcholu	326	260.6	-14.0	0.8
8	Bcholug	326	260.7	-14.0	0.7
9	Bgaussg	279	226.9	-13.8	0.6
10	Bgauss	279	227.4	-13.9	0.6
11	Binv	275	228.6	-13.9	0.7
12	Bihess	270	199.6	-14.2	0.5
1	Dconjp	319	279.5	-14.1	0.8
2	Dconjm	317	281.4	-14.1	0.8
3	Dcholu	308	332.7	-14.1	1.0
4	Dconjptu	302	293.0	-14.0	0.9
5	Dconjmtu	302	293.6	-14.0	0.9
6	Dconjmt	302	293.6	-14.1	0.9
7	Dconjpt	302	293.8	-14.1	0.9
8	Dinv	274	281.4	-13.7	0.8
9	Dgauss	272	271.9	-13.7	0.8
10	Dihess	253	229.8	-14.3	0.6

Table 5. Strong line search and varying second order precision.

Strong line search. The number of successfully solved test problems ranged from 334 for Bconjp and Bconjm down to 253 for Dihess. The mean number of function evaluations ranged from 199.6 for Bihess through to 332.7 for Dcholu. The mean number of function evaluations required by the BFGS implementations was 230 ± 31 compared to 281 ± 52 for the DFP implementations. The mean execution times ranged from 0.6 to 1.0 seconds per test problem.

Weak line search. The number of successfully solved test problems ranged from 329 for Bconjp and Bconjm down to 143 for Dihess. The mean number of function evaluations ranged from 105.4 for Bihess through to 14320.2 for Dcholu. The mean number of function evaluations required by the BFGS implementations was 119 ± 14 compared to 11820 ± 2505 for the DFP implementations. The mean execution times ranged from 0.4 to 49.6 seconds per test problem. Once again, with the weak line search the BFGS implementations required far fewer function evaluations than the DFP implementations and about half the number of function evaluations of the BFGS implementations with the strong line search.

Rank	*Method*	*Succ*	*Fcnt*	*Accy*	*Time*
1	Bconjm	329	114.5	-10.5	0.4
2	Bconjp	329	117.9	-10.6	0.4
3	Bcholug	321	132.6	-10.6	0.5
4	Bcholu	320	132.1	-10.5	0.5
5	Bconjmt	317	117.5	-10.4	0.5
6	Bconjptu	316	116.0	-10.4	0.4
7	Bconjpt	316	117.2	-10.4	0.5
8	Bconjmtu	315	119.2	-10.5	0.4
9	Binv	275	111.1	-10.4	0.4
10	Bgaussg	275	111.8	-10.5	0.4
11	Bgauss	275	112.0	-10.5	0.4
12	Bihess	263	105.4	-10.4	0.3
1	Dconjp	244	11349.1	-9.9	33.1
2	Dconjm	233	11340.5	-9.9	32.8
3	Dcholu	230	14320.2	-9.8	49.6
4	Dconjmt	204	10388.5	-9.4	43.7
5	Dconjmtu	204	11142.6	-9.4	40.2
6	Dconjpt	203	12040.4	-9.3	48.5
7	Dconjptu	202	10680.1	-9.4	38.3
8	Dinv	161	11777.2	-9.7	41.4
9	Dgauss	156	11222.3	-9.6	36.6
10	Dihess	143	9316.5	-8.6	25.9

Table 6. Weak line search and varying second order precision.

3.3. **Comparisons.** Due to the similarity of several of the implementations some comparisons are made which reduce the number of implementations considered in the following sections.

Direct updates. The Binv, Bgauss and Bgaussg implementations performed very similarly regardless of the line search strength or the precision of the approximate Hessian information. This also applies to the DFP implementations Dinv and Dgauss. As such future comparisons will only use the Binv and Dinv implementations. Note that these implementations would not be chosen in practice as they are more computationally expensive and considered to be less numerically stable than the other direct update implementations considered here. They are included because they performed almost identically to the other implementations and to provide a guideline for the worst performance that would be expected from these implementations. However it is interesting to note that Dinv outperformed Dgauss with each of the line searches and the improved performance of Dinv compared to Dgauss was even more noticeable with the weak line search (Table 6).

Rank	*Method*	*Succ*	*Fcnt*	*Accy*	*Time*
1	Bconjp	334	222.6	-14.1	0.6
2	Bcholu	326	260.6	-14.0	0.8
3	Binv	275	228.6	-13.9	0.7
4	Bihess	270	199.6	-14.2	0.6
1	Dconjp	319	279.5	-14.1	0.8
2	Dcholu	308	332.7	-14.1	1.0
3	Dinv	274	281.4	-13.7	0.8
4	Dihess	253	229.8	-14.3	0.6

Table 7. Strong line search.

Rank	*Method*	*Succ*	*Fcnt*	*Accy*	*Time*
1	Bconjp	329	117.9	-10.6	0.4
2	Bcholu	320	132.1	-10.5	0.5
3	Binv	275	111.1	-10.4	0.4
4	Bihess	263	105.4	-10.4	0.4
1	Dconjp	244	11349.1	-9.9	33.1
2	Dcholu	230	14320.2	-9.8	49.6
3	Dinv	161	11777.2	-9.7	41.4
4	Dihess	143	9316.5	-8.6	25.9

Table 8. Weak line search.

Cholesky factorisations. The performance of the BFGS Cholesky factorisation implementations Bcholu and Bcholug were very similar for each of the line searches. Future comparisons will only use the Bcholu implementation (chosen in part for consistency with the DFP implementation which has no Dcholug counterpart).

Conjugate factorisations. Bconjp and Dconjp were chosen as the best overall performing conjugate factorisation implementations.

Results summary. A reduced set of results which compares the chosen conjugate factorisations with Bihess, Binv and Bcholu for the BFGS implementations and Dihess, Dinv and Dcholu for the DFP implementations as approximate Hessian information varied from 16 to two digits is presented in Tables 7 and 8.

Figures 1–4 show the number of successfully solved test problems for the methods in Tables 7 and 8 as the precision of the approximate Hessian information varies from 16 to two digits. Figures 1 and 2 show the performance of the BFGS methods in Tables 7 and 8 with the strong and weak line searches. Figures 3 and 4 show the performance of the DFP methods. The performance of the DFP methods with the strong line search is similar to the performance of the BFGS methods with both the strong and weak line search.

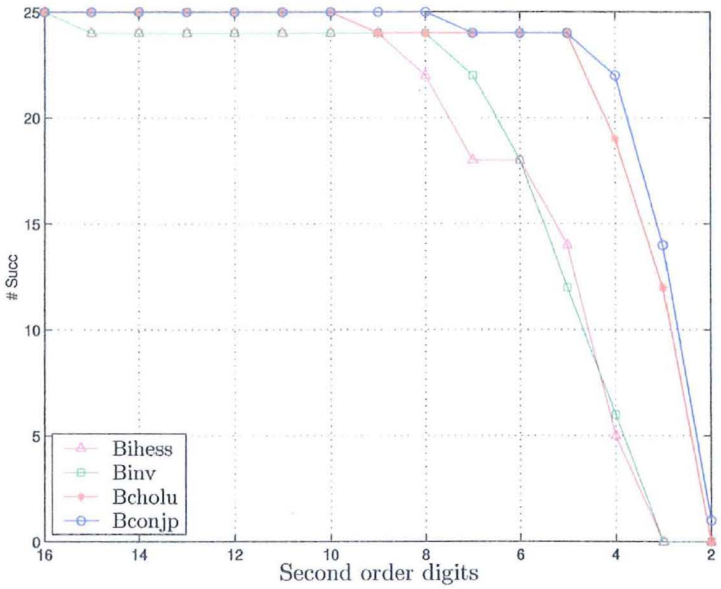

Figure 1. Selected BFGS methods and the strong line search with varying second order precision.

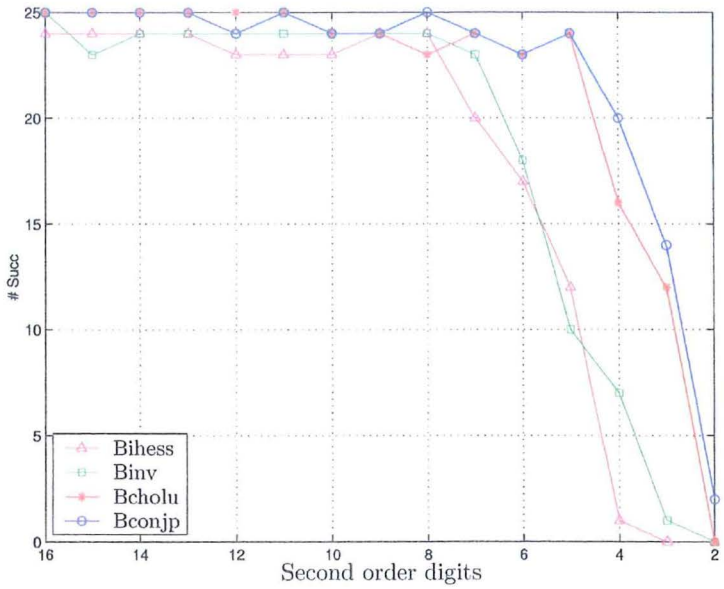

Figure 2. Selected BFGS methods and the weak line search with varying second order precision.

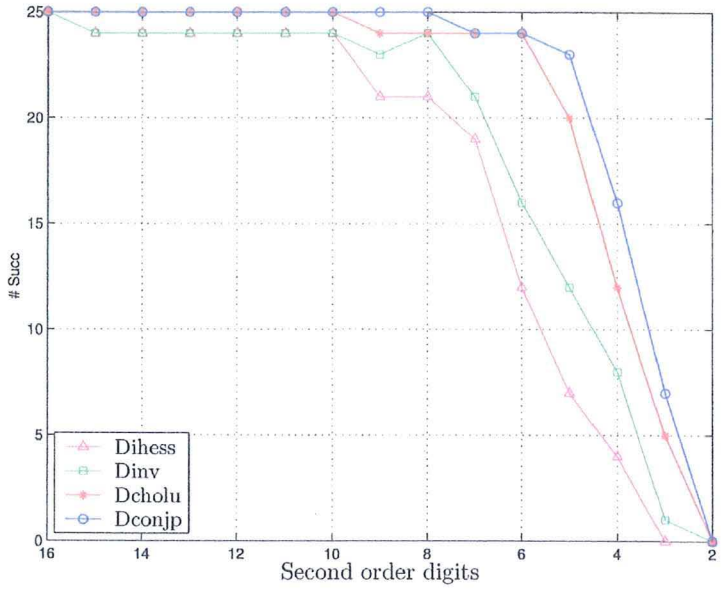

Figure 3. Selected DFP methods and the strong line search with varying second order precision.

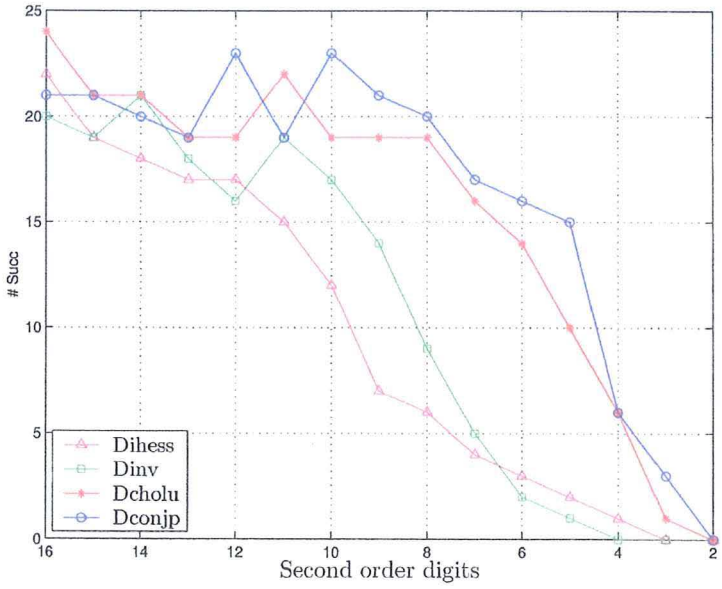

Figure 4. Selected DFP methods and the weak line search with varying second order precision.

3.4. **Quadratic termination.** For any member of the Broyden family of quasi-Newton methods $B_{n+1} = G$ for any n-dimensional quadratic function with Hessian matrix G when exact line searches are used [13, pp. 64–65]. Although it is not possible to carry out exact line searches in practice, this result can be used to see how closely each of the above implementations get to the actual Hessian matrix after $n+1$ iterations. Figure 5 shows $\log_{10} \|H_{n+1} - G^{-1}\|_F$ for each of the BFGS implementations; Binv, Bcholu and Bconjp with the four dimensional Hilbert quadratic and an accurate line search ($\sigma = 10^{-10}$). Note that $\|\cdot\|_F$ represents the Frobenius norm. The difference in norm of the inverse Hessian rather than the Hessian has been used as the inverse Hessian allows the direct calculation of the search direction, whereas a system of equations must be solved if the Hessian is used. Note also that the inverse Hessian is exact but the approximate inverse Hessian matrices H_k are truncated depending on the level of second order precision.

The results for Bihess clutter the figure somewhat and have been omitted. However if included, the plot for Bihess would oscillate between the lines for Bcholu and Bconjp. As mentioned previously, when Bihess is successful it seems to work very well, which is reinforced by these results. The DFP implementations are not shown as they produce almost identical iterates with accurate line searches.

Note that as the precision of the second order information falls below about five digits there is a plateau in Figure 5 with a height of about four. The height of this plateau coincides with the norm of the inverse Hessian of the four dimensional Hilbert quadratic ($\log_{10} \|G^{-1}\|_F \approx 4.0146$). Presumably once the precision of the second order information falls below a certain level there is insufficient information to approximate the inverse Hessian to any significant level. Similar results are produced with Hilbert quadratics of different dimensions. In higher dimensions the height of the plateau matches the norm of the inverse Hessian but the plateau starts at higher levels of second order precision. In lower dimensions the plateau effect is lost and the differences in the performances of these implementations are reduced.

4. Discussion and summary

The performance of 12 BFGS and 10 DFP quasi-Newton implementations on a suite of 25 test functions with two line searches (strong and weak) as the precision of second order information varied from 16 to two

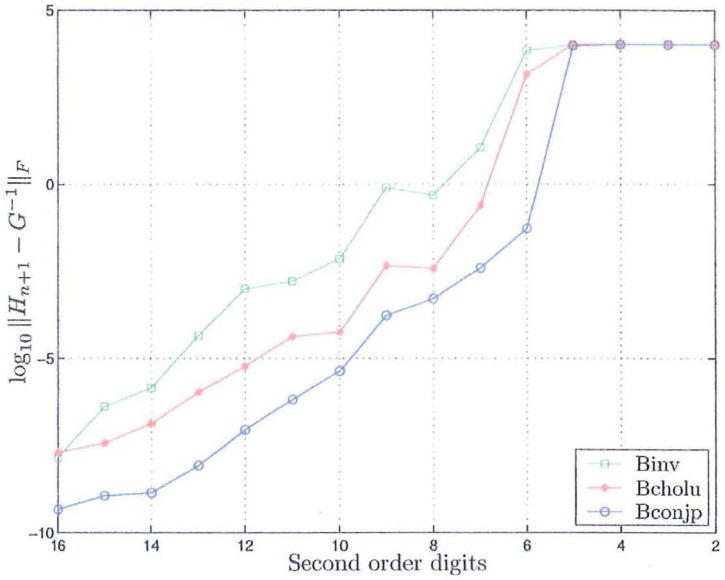

Figure 5. Difference in norm of approximate inverse Hessian with exact inverse Hessian after $n+1$ iterations with varying second order precision.

digits have been presented. Although the BFGS and DFP implementations were quite similar for the strong line search, the BFGS implementations required fewer function evaluations and successfully solved more test problems than the DFP implementations. With the weak line search however, the performance of the BFGS implementations was vastly superior to that of the DFP implementations. Although the BFGS implementations with the strong line search were slightly more robust than the BFGS implementations with the weak line search they required nearly double the number of function evaluations.

When second order information is available to double precision (16 digits) there is no real advantage in any particular implementation. If second order information is only available to single precision (8 digits) then a factorisation strategy greatly improves the performance of the DFP implementations with the weak line search, but does not greatly alter the performance of the BFGS implementations (with either line search) or the DFP implementations with the strong line search.

If second order information is available to reasonable precision then the straightforward inverse Hessian update of the BFGS method, Bihess, produced results which are at least as accurate as those of any

of the other methods considered and requires, on average, fewer function evaluations. However this implementation does not allow for the easy detection of loss of positive definiteness of the inverse approximate Hessian matrices.

Cholesky factor and triangular conjugate factor implementations enable second order information to be updated and a new search direction computed in $O(n^2)$ operations per iteration as well as allowing the easy detection of loss of positive definiteness of the second order matrices. However there is a noticeable drop in performance of the triangular conjugate factor implementations compared to the more straightforward (non-triangularised) conjugate factor implementations. The triangularised conjugate factor implementations perform very similarly to the Cholesky factor implementations. This is not surprising as triangular conjugate factors are also a type of Cholesky factor. Although purely conjecture at this stage, the straightforward conjugate factor implementations probably perform so well because they preserve some useful second order information at each iteration, maybe only making small changes to some of the columns of the factor matrices. This would explain the reduction in performance when triangularised conjugate factors are used — the triangularisation process destroys this information by distributing it across the columns of the triangular factors.

Although the use of modified Cholesky factors allows the easy detection of loss of positive definiteness of the approximate Hessian matrices, a conjugate factorisation eliminates completely the possibility of negative definiteness or indefiniteness of the inverse approximate Hessian matrices whilst maintaining $O(n^2)$ operations efficiency at each iteration.

Figures 1–5 and Tables 5–6 clearly show the importance of a factorisation strategy as the precision of second order information is reduced. The straightforward conjugate factorisation implementation Bconjp successfully solved significantly more test problems with significantly fewer function evaluations than any of the other implementations presented here, including the Cholesky factorisation implementation Bcholu. Also the conjugate factorisation implementation Bconjp produced better approximations to the inverse Hessian matrices of n-dimensional Hilbert quadratics when terminated after $n + 1$ iterations than the other methods. Furthermore, as the precision of the second order information was reduced Bconjp was able to maintain accurate approximations to the inverse Hessian longer than the other methods.

Finally, it is shown in [6] that grids based on conjugate directions have useful practical and theoretical properties, as such conjugate factorisations should also be of practical importance in a wider optimisation context.

5. Acknowlegement

The authors would like to thank Michelle Dalrymple and Robin Turner from the Department of Mathematics and Statistics at the University of Canterbury for their suggestions on various statistical aspects of this paper.

References

[1] K. W. Brodlie, A. R. Gourlay, and J. Greenstadt. Rank-one and rank-two corrections to positive definite matrices expressed in product form. *Institute of Mathematics and its Applications Journal*, 11(1):73–82, February 1973.

[2] C. G. Broyden. Quasi-Newton methods and their application to function minimisation. *Mathematics of Computation*, 21(99):368–381, July 1967.

[3] C. G. Broyden. The convergence of a class of double-rank minimization algorithms. Part 1: General considerations. *Institute of Mathematics and its Applications Journal*, 6(1):76–90, March 1970.

[4] C. G. Broyden. The convergence of a class of double-rank minimization algorithms. Part 2: The new algorithm. *Institute of Mathematics and its Applications Journal*, 6(3):222–231, September 1970.

[5] D. Byatt. Convergent variants of the Nelder-Mead algorithm. Master's thesis, University of Canterbury, Christchurch, New Zealand, June 2000.

[6] D. Byatt, I. D. Coope, and C. J. Price. Conjugate grids for unconstrained optimisation. Research report UCDMS2002/10, University of Canterbury, Christchurch, New Zealand, August 2002.

[7] I. D. Coope. A conjugate direction implementation of the BFGS algorithm with automatic scaling. *Journal of the Australian Mathematics Society Series B*, 31:122–134, 1989.

[8] W. C. Davidon. Variable matric method for minimization. *SIAM Journal on Optimization*, 1(1):1–17, February 1991. Originally published without the preface as Argonne National Laboratory Research and Development Report 5990, May 1959.

[9] L. C. W. Dixon. Quasi-Newton algorithms generate identical points. *Mathematical Programming*, 2:383–387, 1972.

[10] L. C. W. Dixon. Quasi-Newton techniques generate identical points II: The proofs of four new theorems. *Mathematical Programming*, 3(3):345–358, 1972.

[11] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, August 1970.

[12] R. Fletcher. *Practical methods of optimization.* John Wiley & Sons, New York, second edition, 1987.

[13] R. Fletcher and M. J. D. Powell. A rapidly convergent descent method for minimization. *The Computer Journal*, 6(2):163–168, July 1963.

[14] P. E. Gill and W. Murray. Quasi-Newton methods for unconstrained optimization. *Institute of Mathematics and its Applications Journal*, 9(1):91–108, 1972.

[15] P. E. Gill and W. Murray. Modification of matrix factorizations after a rank one update. In D. Jacobs, editor, *The state of the art in numerical analysis*, pages 55–83, London, 1976. Institute of Mathematics and its Applications, Academic Press.

[16] P. E. Gill, W. Murray, and M. H. Wright. *Practical optimization.* Academic Press, London, 1981.

[17] D. Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, January 1970.

[18] L. Grandinetti. Factorization versus non-factorization in quasi-Newtonian algorithms for differentiable optimization. In *Third Symposium on Operations Research (University of Mannheim, Mannheim, 1978)*, pages 255–274, Konigstein, 1979. Hain. Section I.

[19] L. Grandinetti. Factorized variable metric algorithms for unconstrained optimization. In K. Iracki, K. Malanowski, and S. Walukiewicz, editors, *Optimization Techniques. Proceedings of the 9th IFIP Conference on Optimization Techniques, Warsaw, September 4–8, 1979*, volume 23 of *Lecture Notes in Control and Information Sciences*, pages 52–61, Berlin, September 1979. Springer-Verlag. Part 2.

[20] J. Greenstadt. Variations on variable-metric methods. *Mathematics of Computation*, 24(109):1–22, January 1970.

[21] Harwell subroutine library, release 10, Advanced Computing Department, AEA Industrial Technology, Harwell Laboratory, UK, 1990.

[22] MATLAB, The MathWorks Inc., 24 Prime Park Way, Natick, Massachusetts, USA. `http://www.mathworks.com`.

[23] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, March 1981.

[24] D. D. Morrison. Optimization by least squares. *SIAM Journal on Numerical Analysis*, 5:83–88, 1968.

[25] M. J. D. Powell. Updating conjugate directions by the BFGS formula. *Mathematical Programming*, 38(1):29–46, 1987.

[26] R. B. Schnabel and E. Eskow. A revised modified Cholesky factorization algorithm. *SIAM Journal on Optimization*, 9(4):1135–1148, 1999.

[27] D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, July 1970.

[28] A. H. Sherman. On Newton iterative methods for the solution of systems of nonlinear equations. *SIAM Journal on Numerical Analysis*, 15:755–771, 1978.

[29] M. A. Woodbury. Inverting modified matrices. Statistical Research Group Report 42, Princeton University, Princeton, NJ, USA, 1950.

Department of Mathematics and Statistics, University of Canterbury, Private Bag 4800, Christchurch, New Zealand

E-mail address: `d.byatt@math.canterbury.ac.nz`

E-mail address: `i.coope@math.canterbury.ac.nz`

E-mail address: `c.price@math.canterbury.ac.nz`